IAR SYSTEMS
EMBEDDED WORKBENCH
EW 6811

EMBEDDED WORKBENCH
- Runs under Windows 95, NT and 3.1x
- Total integration of compiler, assembler, linker and debugger
- Plug-in architecture for several IAR toolsets
- Hierarchical project presentation
- Tool options configurable on build target, group of files and file level
- Extensive on-line help

C COMPILER
- Fully ANSI C compatible
- Chip-specific extensions to suit development for embedded applications
- Built-in 68HC11-specific optimizer
- Reentrant code
- Supports the entire family of 68HC11 derivatives including the K4 series
- Supported by most popular emulators

ASSEMBLER
- New greatly enhanced assembler
- C Compiler preprocessor integrated with the assembler

C-SPY
- C source and assembler level language debugger
- Powerful handling of complex breakpoints
- C-like macro language
- I/O simulation
- Interrupt simulation

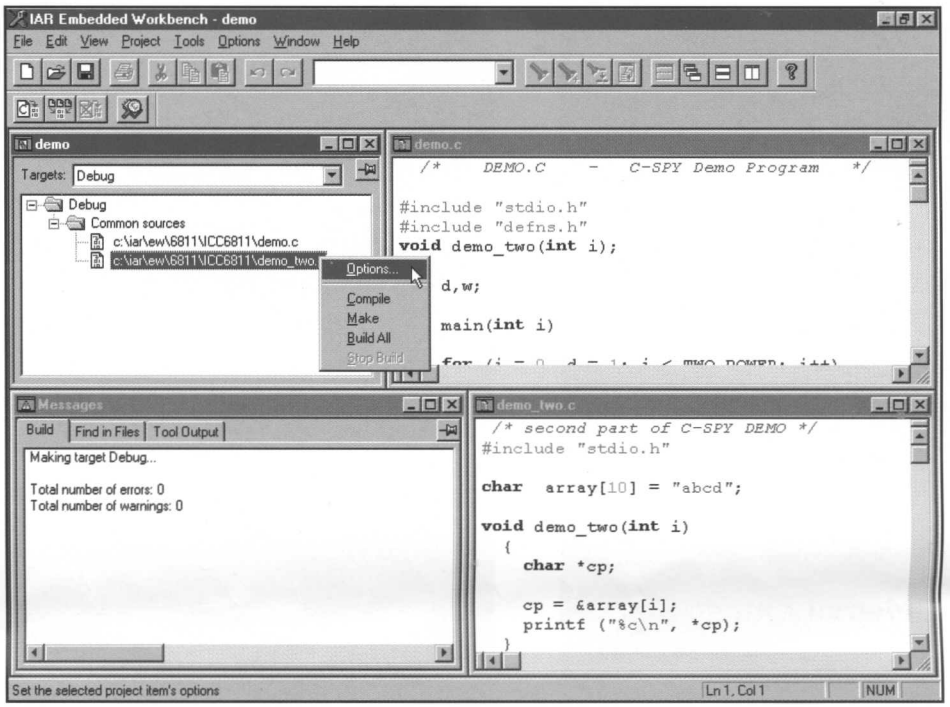
IAR SYSTEMS

The complete development tool for Motorola 68HC11

The IAR Embedded Workbench is a highly evolved development tool for programming embedded applications. The tool offers C to all 68HC11 applications, from single-chip to banked design. The Embedded Workbench combines powerful and efficient optimization with ease-of-use and gives "user friendly" a whole new meaning for programming embedded applications. With its built-in architecture-specific optimizer, the compiler generates very efficient, fast and reliable PROMable code for the entire family of 68HC11 derivatives. Furthermore, in addition to this solid technology, IAR also provides professional technical support, which is yet another reason why engineers adopt IAR C Compilers.

An Integrated Environment for Embedded Project Development

The new Embedded Workbench takes full advantage of the 32-bit Windows 95 and NT environment. For example, it allows compiling to be done in the background. The toolset can also run under Windows 3.1x with the Win32s subsystem (included in the package).



The IAR Embedded Workbench for the 68HC11.

Everything you need for programming embedded applications

The EW6811 integrates the IAR C Compiler, linker, librarian, and assembler in a seamless environment with an easy-to-use project feature and option handling. Each new IAR Systems toolset for other chip families is easily integrated into the Embedded Workbench, reducing the start-up time for new targets by providing a well known programming environment.

Hierarchical project presentation

The project maintenance feature allows organization of projects in a hierarchical tree structure showing the dependency between files. At the highest level of the structure, the different target versions of the application are specified. This makes it possible to have several targets, such as a release target and a debug target, with different settings of options. Each target is built up of one or several groups, which in turn are built up of one or several source files. Options for the compiler and the assembler can be set on target levels, group levels or file levels, whichever is appropriate.

The Make system

The Make system automatically generates a dependency list of output files, source files and even include files. This allows the Make system to only recompile or reassemble the updated parts of the source code, which speeds up the building process.

The Editor

The Editor in EW6811 is a versatile and powerful tool. It offers flexibility in terms of a customizable toolbar and user-defined shortcut keys (not only limited to the Editor). The Editor implements the basic Windows editing commands as well as extensions for C programming, such as C syntax highlighting, and direct jumps to context from error listing.

Extensive on-line help

The on-line help function makes it easy to quickly find specific help about the tool without leaving the Embedded Workbench, which reduces your learning time and therefore also your time to market.

DOS user interface

The IAR C Compiler is also available in a command line version for DOS and UNIX. The DOS version comes with a mouse-controlled, menu-driven user interface allowing all development steps to be performed in an integrated DOS environment.

A Complete Package including C Compiler, Assembler, Linker, Librarian and Run-Time Libraries

The IAR C Compiler, the core product in the EW6811 and the ICC6811, is fully compatible with the ANSI C standard. All data types required by ANSI are supported without exception (*see Figure 1*). Full ANSI C compatibility also means that the compiler conforms to all requirements placed by ANSI on run-time behavior, even those less known yet important requirements such as integral promotion and precision in calculating floating point.

Float and *double* are represented in the IEEE 32-bit and 64-bit precision. *struct, array, union, enum,* and *bitfield* are also supported.

68HC11 K4 specific support

The IAR C Compiler supports the entire family of 68HC11 derivatives including the 68HC11 K4 MMU.

Memory models for any hardware design

Every design has its own memory requirement. The compiler has three different memory models to allow a best fit selection. Depending on the application, the appropriate memory model can be selected (*see Figure 2*).

The small memory model makes the compiler use the short (direct) addressing mode when accessing static/global variables. It is well-suited for single-chip design.

The large memory model is used for larger applications and supports external RAM. The entire 64K space can be split between CODE and DATA.

The banked memory model extends the code area beyond 64K and up to 4MB with the help of a transparent bank-switching routine supplied in source form.

DATA TYPE	SIZE (bytes)	VALUE RANGE
signed char	1	-128 to +127
unsigned char	1	0 to 255
signed short	2	-32768 to +32767
unsigned short	2	0 to 65535
signed int	2	-32768 to +32767
unsigned int	2	0 to 65535
signed long	4	-2^{31} to $2^{31}-1$
unsigned long	4	0 to $2^{32}-1$
float IEEE 32-bit	4	±1.18E-38 to ±3.39E+38, 7 digits
double IEEE 64-bit	4/8	±2.23E-308 to ±1.79E+308,16 digits
data pointer	1/2	object address
function pointer	2/3	function address

Figure 1. *Data representation supported by the IAR C Compiler.*

Reentrancy

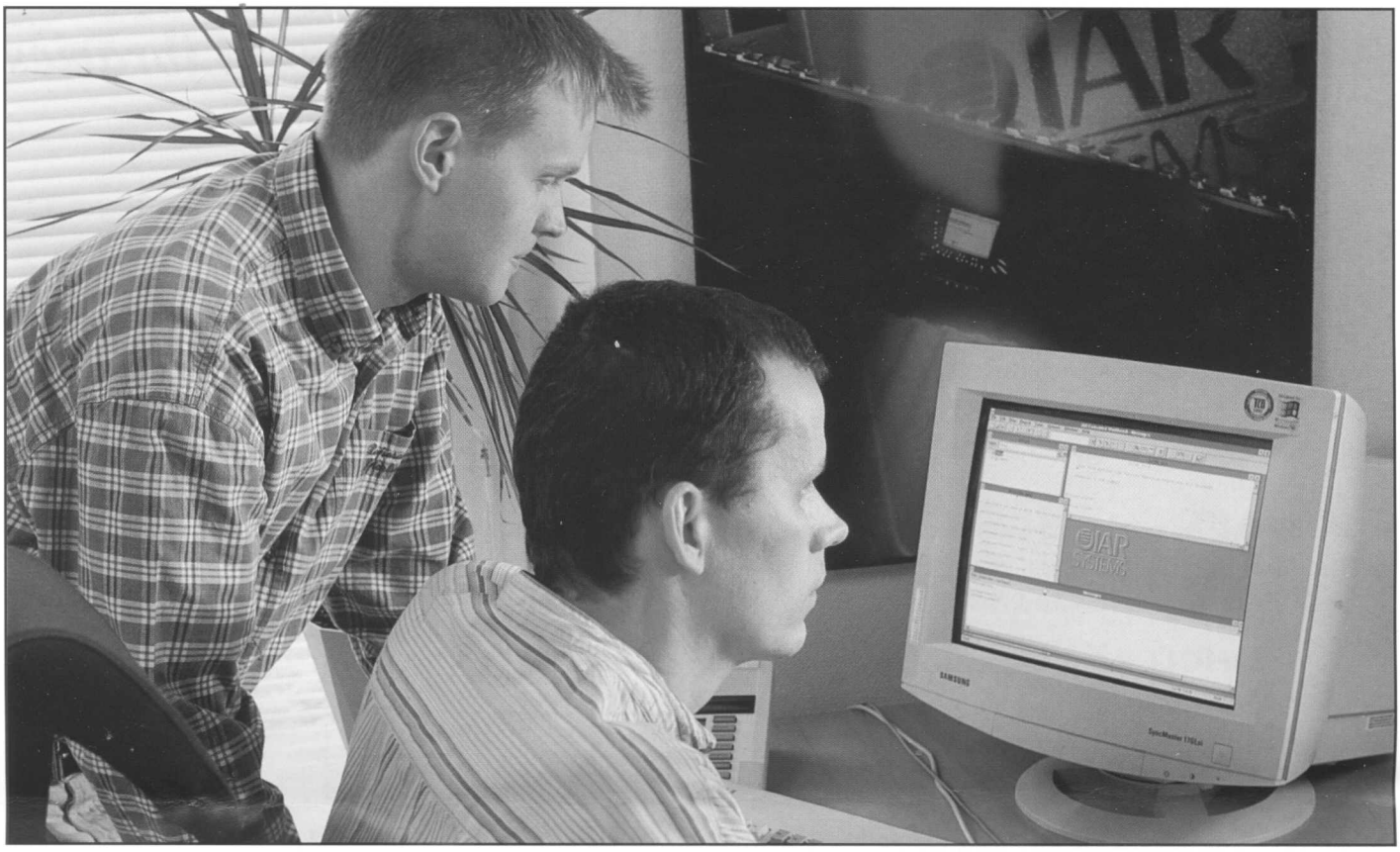
The compiler generates fully reentrant code. Any function can be interrupted and called from the interrupting routine without the risk of corrupting the local environment of the function. This feature makes the IAR C Compiler ideal to use with real time operating systems.

Built-in 68HC11 specific optimizer

The outstanding and powerful optimization technique, together with a reliable high quality program, makes the IAR C Compiler unique. The built-in optimizer will automatically generate efficient and compact code for the 68HC11 architecture (*see Figure 3*).

MEMORY	CODE AREA	DEFAULT CODE POINTER	DATA AREA	DEFAULT DATA POINTER	STORAGE CLASS
SMALL	<64KB	Non-banked	256 Bytes*	npage	zpage
LARGE	<64KB	Non-banked	<64KB	npage	npage
BANKED	>64KB, <4MB	Banked	<64KB	npage	npage
*zpage extended keyword can be used to overrride default in the small memory model					

Figure 2. *Memory models. The three different memory models allow a best fit selection.*



The user friendly IAR Embedded Workbench is developed specifically for programming embedded applications.

Depending on the application, execution speed may be more critical than smaller code. To meet this need, the compiler has the powerful feature of allowing the user to favor speed optimization over code size. For example, speed optimizations perform larger structure copying in-line. The IAR C Compiler performs safe but aggressive optimizations. Among these, the ability to identify register variables is very useful in applications that utilize pointers heavily. In addition, a combination of a large number of low-level optimizations reduces code size even further.

Absolute read/write at C level

It is possible to access specific memory locations directly from C. The following example shows the access to location 4000H.

```
#define PORT (*(char*)0x4000);
/* pointer to addr 4000H*/
void read_write(char c)
{
PORT=c;


/*writes c to location 4000H*/

c=PORT;
/*reads location 4000H into c*/
}
```

68HC11 specific extensions

To ideally suit development for embedded systems, standard C needs additional functionality. IAR Systems has defined a set of extensions to ANSI C, specific to the 68HC11 architecture (*see Figure 4*). All of these extended keywords can be invoked by using the *#pragma* directive, which maintains compatibility with ANSI C as well as code portability.

EXAMPLES OF OPTIMIZATION TECHNIQUES
Strength reduction – Constant folding – Short/long jump optimization – Case/switch optimization – Register content – Assembler level peephole optimization – Reversing branch conditions – Removing unreachable code – Eliminating duplicate code – Eliminating superfluous branches – TSX reductions – Runtime stack cleaning optimizations – Register/flag live analysis – Cross call – Trading Y for X

Figure 3. *The different optimization techniques that can be used by the built-in optimizer.*

All these extensions, coupled with absolute read/write access, minimize the need for assembly language routines.

Efficient floating point

The compiler comes with full floating point support. It follows the IEEE 32-bit and 64-bit representation using an IAR Systems proprietary algorithm, which makes floating point manipulation extremely fast.

Assembler

The IAR C Compiler kit comes with a new relocatable structured assembler. The assembler, A6811, is written in C and uses the same memory-based principles as our other tools for optimal speed. To avoid memory problems, the assembler incorporates an extended memory handler, which means that only the memory of the host limits the file size. This is also valid for the number of symbols in a module or file. The symbols, internal as well as external, can be used in any kind of expression, a feature that is unique to IAR assemblers.

TYPE	KEYWORD	DESCRIPTION
Function	interrupt	Creates an interrupt function. Inserts the vector table and returns with a RTI.
	monitor	Turns off the interrupts while executing a monitor function.
	banked	Declares a banked function
	non_banked	Declares a non-banked function
Variable	no_init	Puts a variable in the no_init segment (battery-backed RAM)
	zpage	Storage and pointer attribute acting on the zero page
	npage	Storage and pointer attribute acting on the 64KB address space
Segment	codeseg	Renames the CODE segment
	constseg	Creates a new segment for constant data
	dataseg	Renames the DATA segment (These are mostly used to place code and data sections in non-consecutive address ranges)
Intrinsic	enable_interrupt	Turns on interrupts (CLI).
	disable_interrupt	Turns off interrupts (SEI).
	wait_for_interrupt	Executes a WAIT instruction (WAI).
	_opc	Inserts the opcode of an instruction into the object code.

Figure 4. *IAR Systems embedded C extensions.*

Macro-assembler for time-critical routines

The assembler provides the option of coding time-critical sections of the application in assembly without losing the advantages of the C language. The preprocessor of the C compiler is incorporated in the assembler, thus allowing use of the full ANSI C macro language, with conditional assembly, macro definitions, if statements, etc. C include files can also be used in an assembly program. All modules written in assembly can easily be accessed from C and vice versa, making the interchange between C and assembly a straightforward process.

Structured assembler

Directives for structured assembly, also called structured control flow, generate assembler source-code for comparison and jump instructions. The generated code is assembled like ordinary code that is similar to macros.

Powerful set of assembler directives

The assembler provides an extensive set of directives to allow total control of code and data segmentation. Directives also allow the creation of multiple modules within a file, macro definitions and variable declarations.

Linker

The IAR XLINK Linker supports complete linking, relocation and format generation to produce 68HC11 PROMable code (*see Figure 5*). XLINK generates over 30 different formats and is compatible with most popular emulators and EPROM burners. XLINK is extremely versatile in allocating any code or data to a start address, and checking for overflow. Detailed cross reference and map listings with segments, symbol information, variable locations, and function addresses are easily generated.

EXAMPLES OF LINKER COMMANDS	DESCRIPTION
-Z seg_def	Allocates a list of segments at a specific address
-F format_name	Selects one of more than 30 different absolute output formats
-x -l file_name	Generates a map file containing the absolute addresses of modules, segments, entry points, global/static variables, and functions
-D symbol=value	Defines a global symbol and equates it to a certain value

Figure 5. *Examples of different linker commands.*

BMW uses IAR C Compiler tools for 68HC11 in the development of door lock systems for their 3, 5, and 7-series.

Librarian

The XLIB Librarian creates and maintains libraries and library modules. Listings of modules, entry points, and symbolic information contained in every library are easily generated. XLIB can also change the attributes in a file or library to be either loaded only if referred to or loaded without being referred to.

ANSI C libraries

The IAR C Compiler kit comes with all libraries required by ANSI. Additionally, it comes with low level routines required for embedded systems development (*see Figure 6*). These routines are provided in source code.

Malloc, realloc, free and heap

A powerful feature uniquely offered by the IAR C Compiler kit is its ability to use malloc and free, plus the flexibility of controlling the heap size. A file called *heap.c* is provided in source code, which allows the user to increase or decrease the heap size.

Full, medium and small printf/scanf

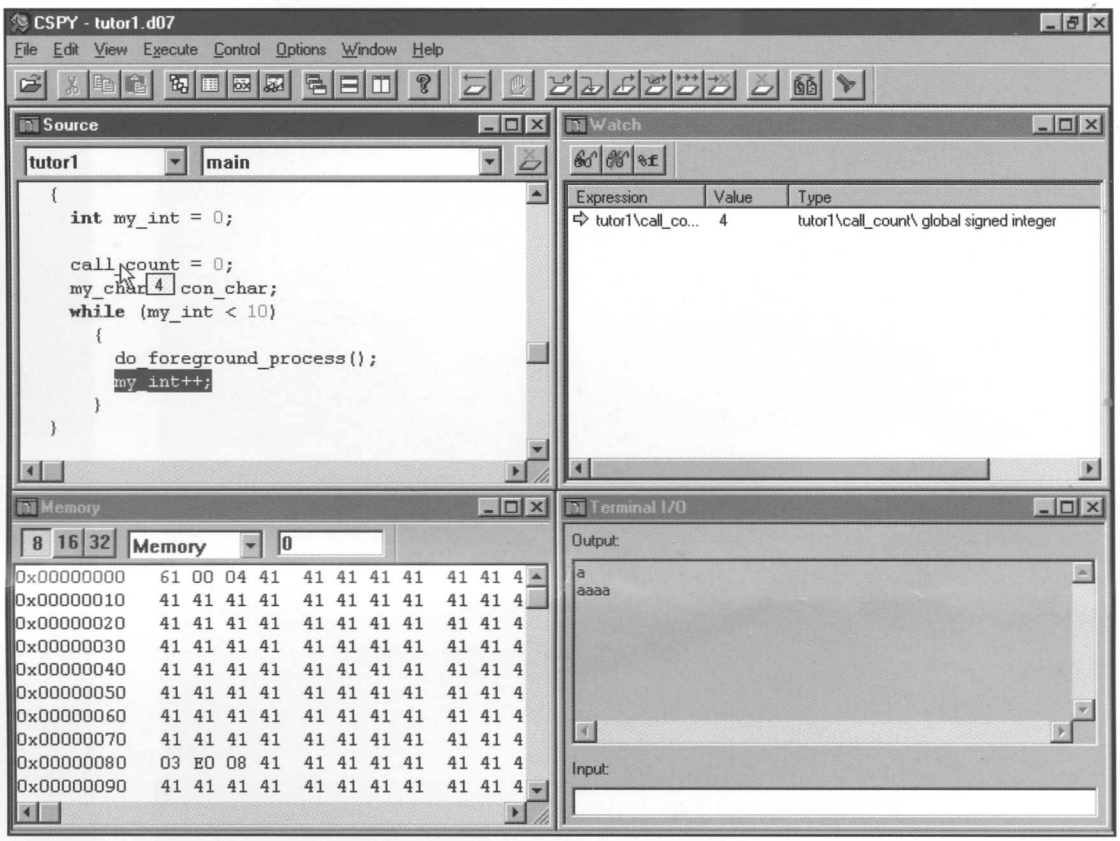
By default, the compiler supports full ANSI *printf, sprintf, scanf,* and *sscanf* functionality including floating point representation. The compiler also includes reduced versions of the *printf* and *scanf* formatters that support only the most commonly used % specifiers. This function reduces code size and increases speed. Small, medium and full *printf* formatters are available.

C LIBRARY FUNCTIONS <name.h>
DIAGNOSTICS <assert.h> assert
CHARACTER HANDLING <ctype.h> isnum, isalnum, isalpha, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace, isupper, isxdigit tolower, toupper
VARIABLE ARGUMENTS <stdarg.h> va_arg, va_end, va_start
NON LOCAL JUMPS <setjmp.h> longjmp, setjmp
INPUT/OUTPUT <stdio.h> getchar, gets, printf, putchar, puts, scanf, sscanf, sprintf
GENERAL UTILITIES <stdlib.h> abort, abs, atof, atoi, bsearch, calloc, div, exit, free, labs, ldiv, malloc, rand, realloc, srand, strtod, strtol, strtoul, qsort
STRING HANDLING <string.h> memchr, memcmp, memcpy, memmove, memset, strcat, strchr, strcmp, strcpy, strcspn, strerror, strlen, strncat, strncmp, strncpy, strpbrk, strrchr, strspn, strstr, strtok, strxfrm
MATHEMATICS <math.h> acos, asin, atan, atan2, ceil, cos, cosh, exp, exp10, fabs, floor, fmod, ldexp, log, log10, modf, pow, sin, sinh, sqrt, tan, tanh
LOW-LEVEL ROUTINES <iccbutl.h> _formatted_write, _formatted_read

Figure 6. *Library functions. IAR C Compiler comes with all libraries required by ANSI.*

More than just a High Level Language Debugger

The IAR C-SPY, CW6811, is a high level language debugger incorporating a complete C expression analyzer and full C-type knowledge. It combines the detailed control of code execution needed for embedded development debugging with the flexibility and power of the C language. CW6811 shows the calling stack as well as tracing on both statement and assembler levels. The source window can display C source code and mix it with assembler. There is also a "locals" window showing the auto variables and parameters for the current function. CW6811 is user-friendly with customizable toolbars, a drag and drop facility as well as user-configurable shortcut keys.



The IAR C-SPY for debugging 68HC11 application.

Powerful breakpoint setting

CW6811 allows you to set an unlimited number of breakpoints. Breakpoints can be set on C statements, assembler instructions, and on any address with an access type of *read*, *write* and *opcode* fetch, or as a combination of these. The breakpoint can be extended with an optional condition. After triggering a breakpoint, any optional macro commands can be executed.

C-like macro language

A powerful, yet easy-to-use C-like macro language can tailor the environment used for debugging in the C-SPY debugger. It includes special system macros for host file I/O simulation, reset, start up, and shut down, as well as statements such as *for* loops, *while* loops, *if* and *return*. Functions declared in the macro can contain specific C-SPY variables, both local and global.

Interrupt simulation

Interrupt simulation implements commands to launch specific interrupts at a specific cycle-count or periodically to a given cycle-interval. The interrupt simulation can also be set to generate sporadic as well as periodic events.

I/O simulation

Breakpoint simulation and the macro language allow the most complex external environment to be simulated. Since I/O simulation is built up with the macro language, it is easy to customize and very easy to extend. CW6811 terminal I/O emulation

C-SPY

offers a console window for target system I/O. This unique feature is useful for debugging embedded applications when logical flows are of interest or the target is not yet ready.

Watch points

The Watch window makes it possible to watch any expression. The window itself will be updated whenever a breakpoint is triggered or a step is finished. Any variable can be modified during the execution by using specific C expressions.

Assembler low level debugger

The source window for the assembler debugger displays the assembler instructions. It has a built-in assembler and disassembler function, menu and register window, and can evaluate assembler expressions.

C-SPY for DOS and UNIX

C-SPY 6811 is also available under DOS and UNIX. The DOS and UNIX versions differ in functionality compared with the Windows version. The main differences are the normal restrictions that the DOS environment offers compared to the Windows environment, the lack of complex breakpoints and limited interrupt simulation.

Optional ROM-monitor interface

The IAR C-SPY debugger is also available in a ROM-monitor version, CW6811R and CS6811R. The evaluation board is connected via the serial port and controlled from the CW/CS6811R. Together with preconfigured monitors for the Motorola EVB, EVS and EVM boards, the CW/CS6811R is an inexpensive and powerful tool for simulating the application in a hardware environment.

Optional emulator interface

C-SPY for 68HC11 is also available as an emulator interface version. Contact IAR Systems for an updated list of supported emulators.

NO LONGER SUPPORTED

Support and Updates

IAR Systems 68HC11 toolset comes with the following benefits:

- Free telephone, fax, and Email technical support.
- 90 days warranty after purchase.
- Extensive documentation including step-by-step tutorials.

Hosts

PC: Minimum 386 with Windows 95, Windows NT 3.5 or Windows 3.1x with at least 4MB RAM available for the EW6811. DOS 4.x with at least 4MB RAM available for the ICC6811.

SUN 4 and Solaris (only the ICC6811 and CS6811 available).

HP 9000 series 700 (only the ICC6811 and CS6811 available).

CONTACT INFORMATION

USA
IAR Systems Software, Inc.
One Maritime Plaza
San Francisco, CA 94111
Phone: +1 415-765-5500
Fax: +1 415-765-5503
Email: info@iar.com

SWEDEN
IAR Systems AB
P.O. Box 23051
S-750 23 Uppsala
Phone: +46 18 16 78 00
Fax: +46 18 16 78 38
Email: info@iar.se

GERMANY
IAR Systems GmbH
Brucknerstrasse 27
D-81677 Munich
Phone: +49 89 470 6022
Fax: +49 89 470 9565
Email: info@iar.de

UK
IAR Systems Ltd
9 Spice Court, Ivory Sq.
London SW11 3UE
Phone: +44 171 924 3334
Fax: +44 171 924 5341
Email: info@iarsys.co.uk

IAR SYSTEMS

http://www.iar.com